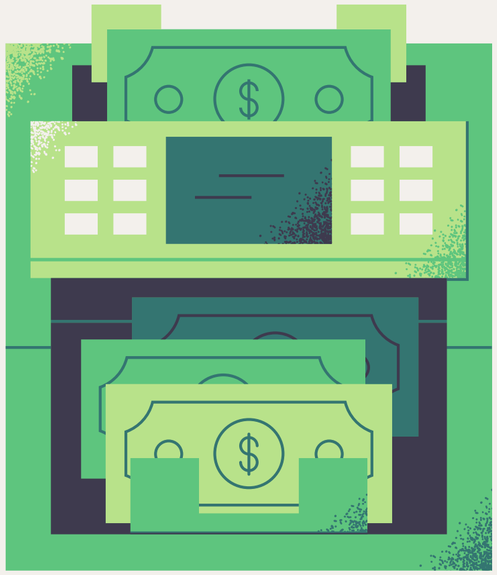
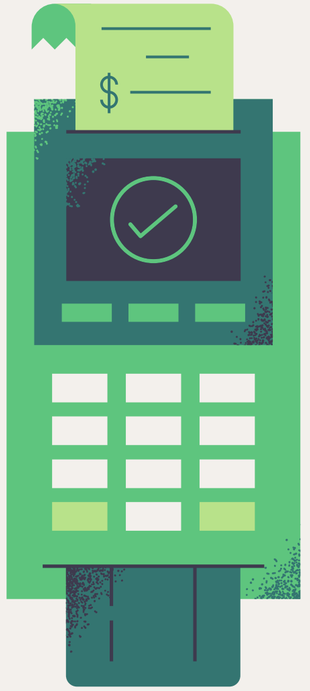
FAKE CURRENCY DETECTION





A V KISHORE KUMAR
CH.EN.U4CSE20134

CONTENTS

PROBLEM STATEMENT

The detection of fake currency is a serious issue for both people and companies. Counterfeiters are constantly developing new techniques and strategies to produce fake banknotes that are almost impossible to tell apart from real money, at least to the human eye. In this report, I'll introduce you to machine learning's use in detecting counterfeit money.

CURRENT LITERATURE SURVEY

-> Used Convolutional NeuralNetworks(CNN), Singleshot multi-box detector(SSD), and Multi-layer perception(MLP) for currency recognition.

-> A Lightweight Three Layers CNN.

-> Generative Adversarial Networks (GANs).

-> A supervised learning model using the dimensional data of currency notes.

-> Deep CNN for both feature extraction and recognition and SVM for classification.

MODEL USED

&

HOW THE EXECUTION IS BEING DONE?

LOGISTIC REGRESSION

RANDOM FOREST CLASSIFIER

SUPPORT VECTOR MACHINE

K NEAREST NEIGHBORS

NAIVE BAYES

K-MEANS(UNSUPERVISED)

- For our objective, we'll can use all algorithms that work well for binary classification.
- Random Forest Classifier: Grows multiple decision trees that are merged together for a more accurate prediction.
- Support Vector Machine: We'll calculate a hyperplane that categorizes the real and fake banknotes.
- K Nearest Neighbours: Finds the distances between a query and training data, selecting the specified number of neighbors (K) closest to the query, then votes for the most frequent label.
- Logistic Regression: Predicts fake/ real by analyzing the relationship between one or more existing independent variables.
- Naive Bayes: Based on applying Bayes' theorem with strong independence assumptions between the features.

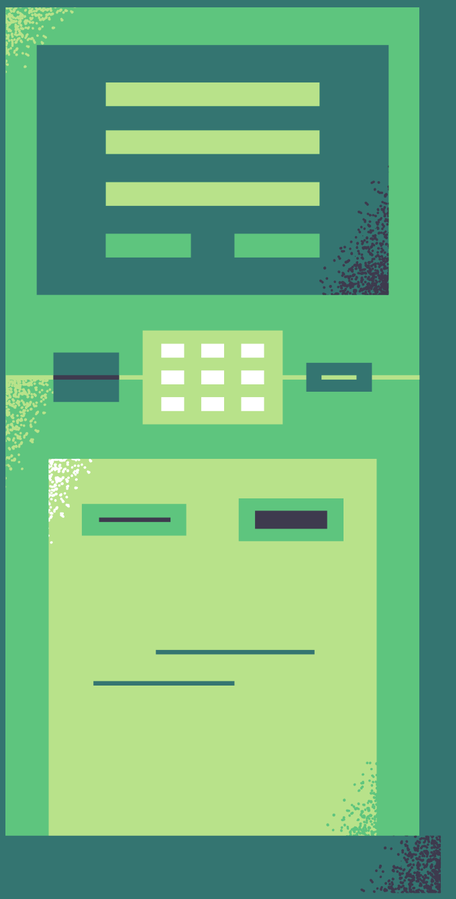
ABOUT THE DATASET

Dataset link:
https://github.com/kishorekumar14/Fake_Currency_Detection/blob/main/bank_notes.csv

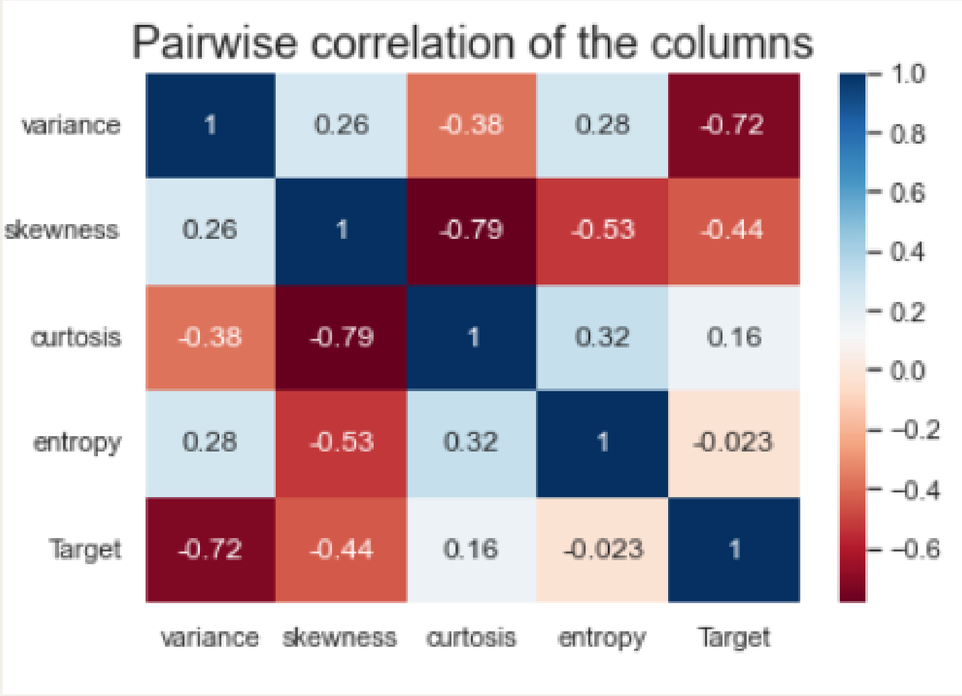
Originally created for detection of counterfeit bank notes consisting of 1372 rows and 5 columns.
For our objective, we'll be using the whole dataset as it is.
The dataset contains four input characterictics:

1. Variance: The variance of the image transformed into wavelets.
2. Skewness: The asymmetry of the image transformed into wavelets.
3. Kurtosis: Kurtosis of the image transformed into wavelets.
4. Entropy: The entropy of the image.

The output target value is 1 for fake banknotes and 0 for real banknotes.



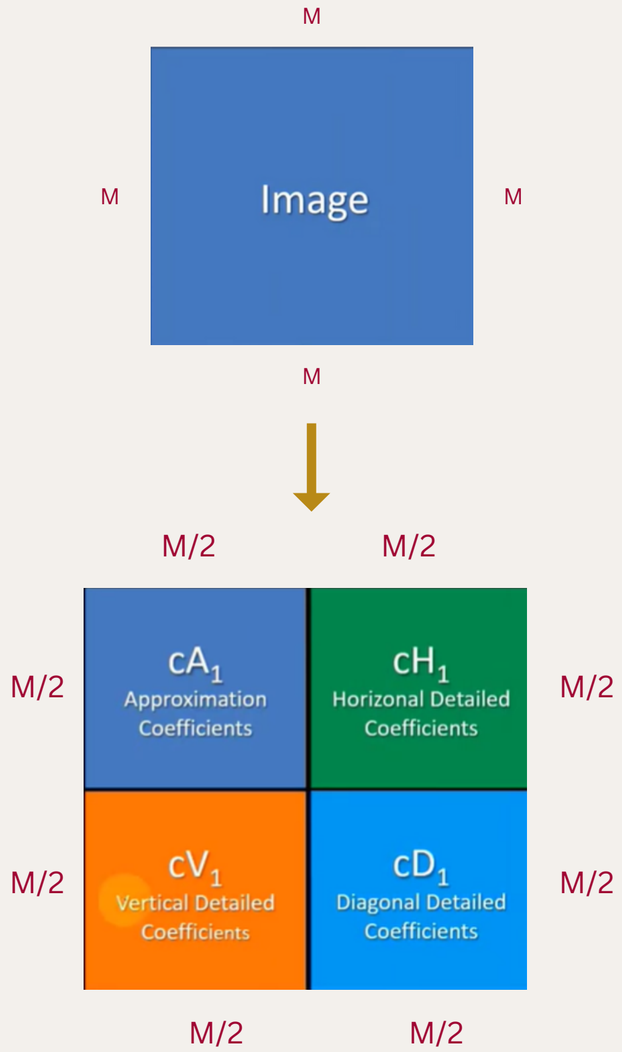
DATA PREPROCESSING AND ANALYZING THE DATASET



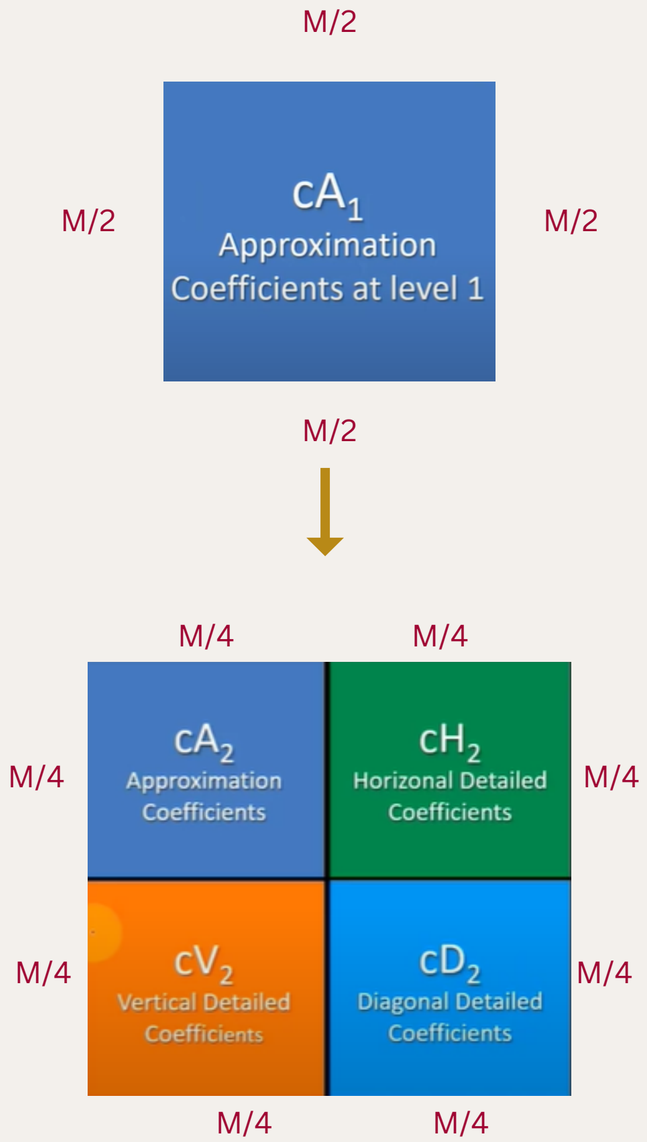
HOW DATA IS BEING TAKEN FROM CURRENCY NOTES:

Image transformed into wavelets:

LEVEL-1



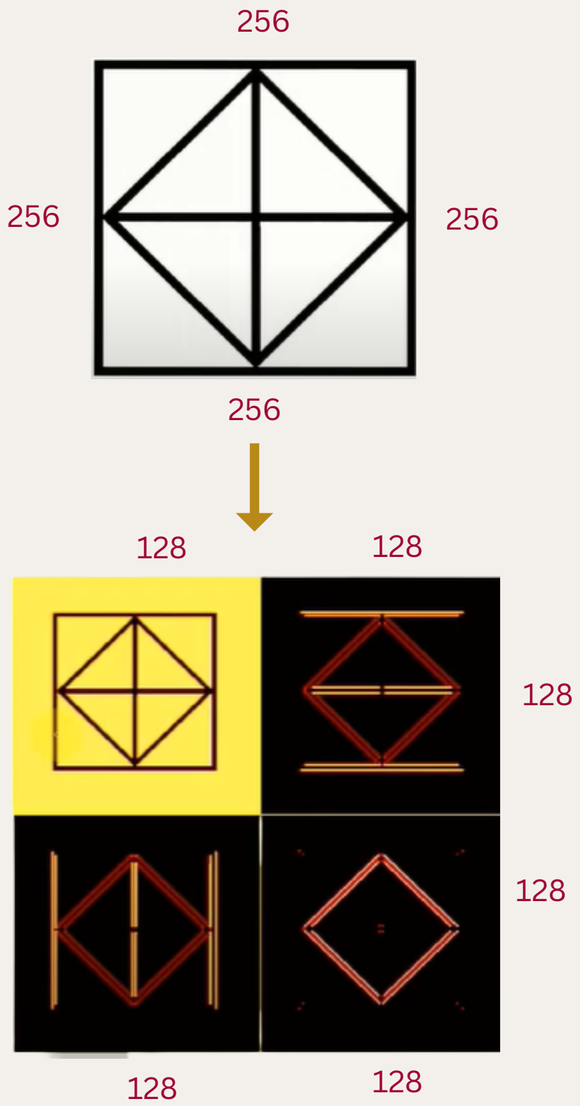
LEVEL-2



HOW DATA IS BEING TAKEN FROM CURRENCY NOTES:

Image transformed into wavelets:

Example 1

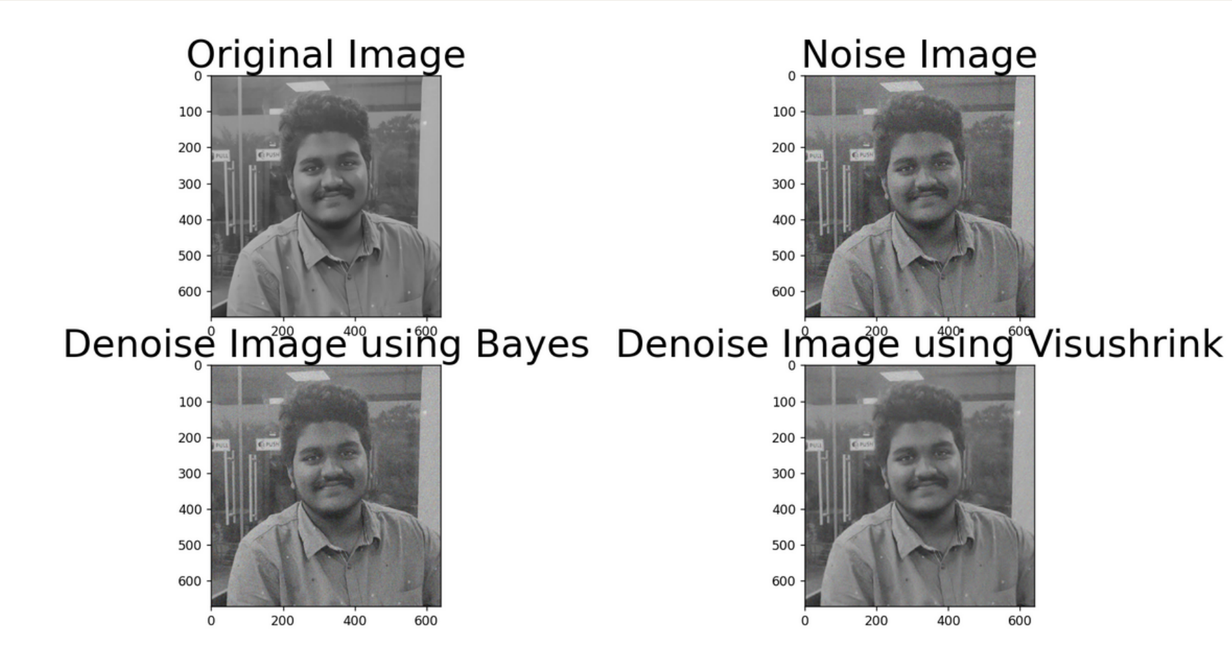


Example 2

HOW DATA IS BEING TAKEN FROM CURRENCY NOTES:

Image transformed into wavelets:

For Greyscale images, Example:



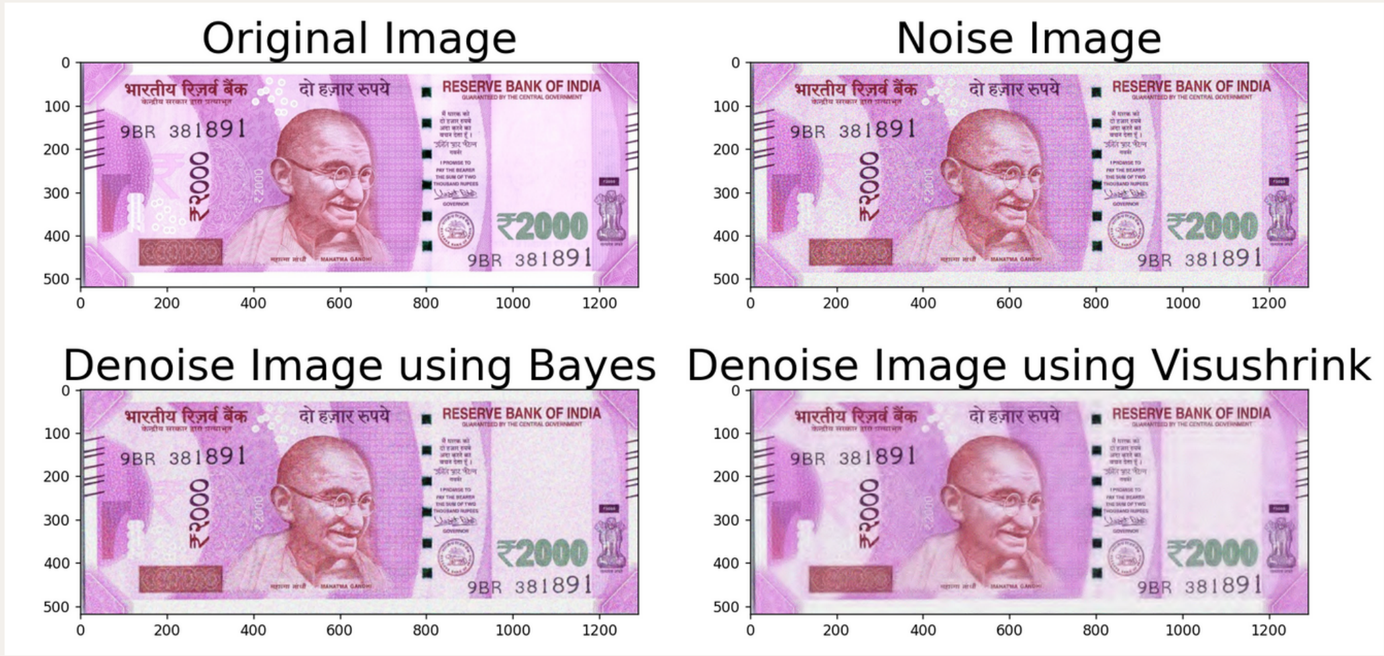
Result from transformation:

```
PSNR [Original Vs. Noisy Image]: 20.002027127528386
PSNR [original vs. denoise[visushrink]]: 26.966336249648776
PSNR [original vs denoise[bayes]]: 22.80923074287522
PS D:\KISHORE\5th sem\Machine Learning\Capstone Project\REVIEWS
```

HOW DATA IS BEING TAKEN FROM CURRENCY NOTES:

Image transformed into wavelets:

For color images, Example:



Result from transformation:

```
PSNR [Original Vs. Noisy Image]: 17.951281615786733
PSNR [original vs. denoise[visushrink]]: 22.71674890233479
PSNR [original vs denoise[bayes]]: 23.54743879285358
```

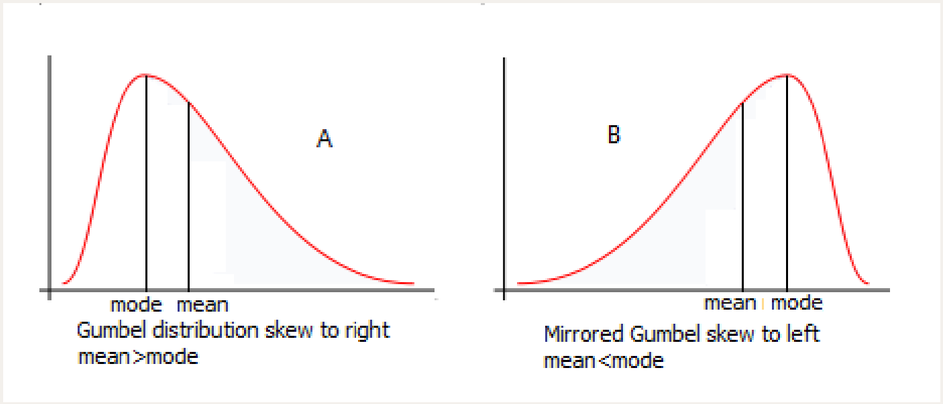
HOW TO CALCULATE VARIANCE, SKEWNESS, KURTOSIS, ENTROPY:

Skewness essentially measures the symmetry of the distribution, while kurtosis determines the heaviness of the distribution tails.

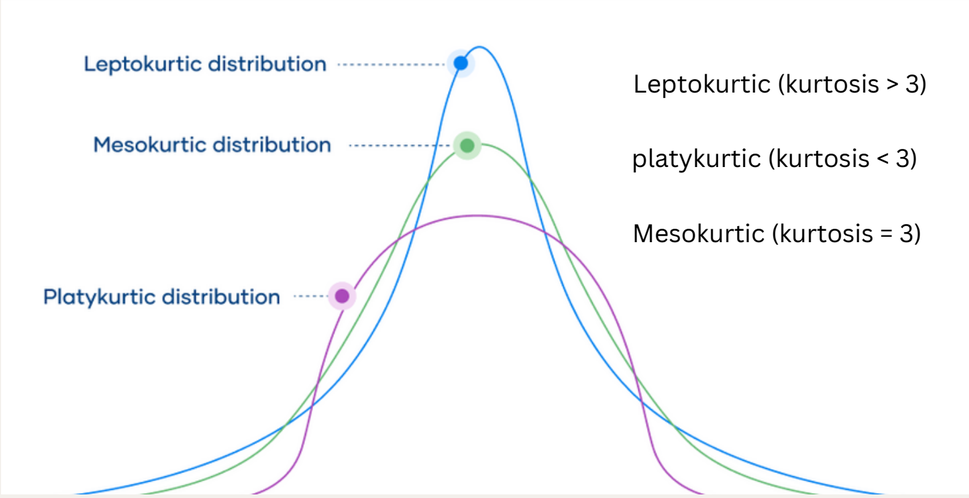
Skewness:

$$\text{Pearson's first coefficient} = \frac{\text{Mean} - \text{Mode}}{\text{Standard Deviation}}$$

$$\text{Pearson's second coefficient} = \frac{3\,(\text{Mean} - \text{Median})}{\text{Standard Deviation}}$$



Kurtosis:



$$\text{Kurtosis} = n * \Sigma^{n}_{i}(Y_i - \bar{Y})^4 \,/\, (\Sigma^{n}_{i}(Y_i - \bar{Y})^2)^2$$

Y_i: ith Variable of the Distribution
$\bar{Y}$: Mean of the Distribution
n: No. of Variables in the Distribution

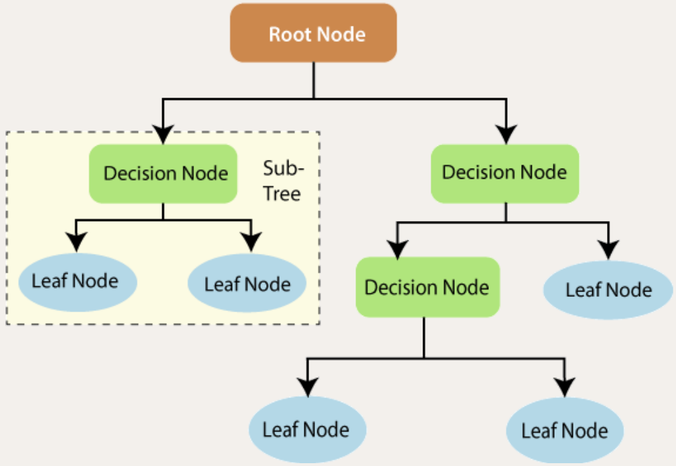
HOW TO CALCULATE VARIANCE, SKEWNESS, KURTOSIS, ENTROPY:

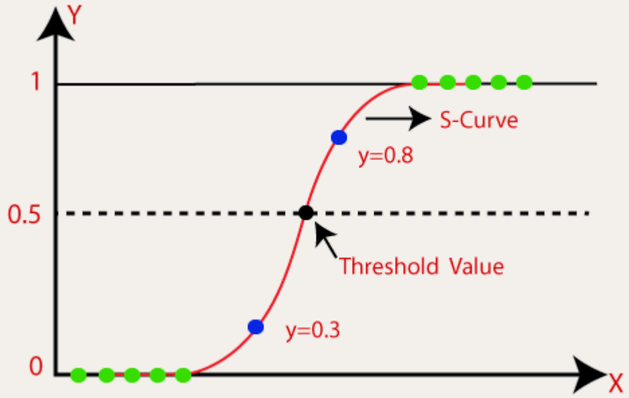
Entropy: The entropy of an image can be calculated by calculating at each pixel position (i,j) the entropy of the pixel-values within a 2-dim region centered at (i,j).

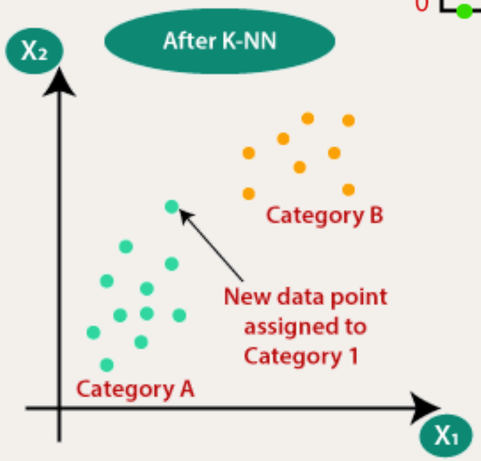
RGB image ---> GreyScale image ---> Entropy image

Variance: Variance measures variability from the average or mean.

variance = sum((x - mean(x)).^2)/(length(x) - 1);







PROPOSED SOLUTION

First, we needed to preprocess the data and convert it into a format that can be used to train machine learning algorithms. Divide the data into Training and Test sets. Since our output variable can take only two values, real note or fake note. I will train and test our model for fake currency detection by using the Logistic Regression Algorithm and further apply more supervised machine learning algorithms and compare them.

WORK-FLOW AND DATA PRE-PROCESSING

- First, I have checked for Null or missing values in the dataset using isnull() function.
- Then, I checked for incorrect values by looking at their datatypes using the function dtypes and info().
- Then, I used duplicated() function to check for duplicate values in the dataset. I got 24 duplicate rows in it. I have deleted the duplicated rows using drop_duplicates().
- Now my dataset is clean and ready to use. I can proceed with splitting my dataset into training and testing with 80% for training and 20% to test.
- Check whether training and testing data are split with a good ratio of real and fake data points.

WORK-FLOW: LOGISTIC REGRESSION

- From sklearn.linear_model we'll import LogisticRegression.
- Create a logistic object using the sigmoid activation function.
- Now fit the training data to logistic object.
- Now, calculate the performance metrics for the testing data.
- Logistic regression is for binary classification only. It computes the probability of an event occurrence.

Linear Regression Equation:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n$$

Where, y is a dependent variable and x1, x2 ... and Xn are explanatory variables.

Sigmoid Function:

$$p = \frac{1}{1 + e^{-y}}$$

Apply Sigmoid function on linear regression:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n)}}$$

KEYWORDS:

sigmoid function= acts as an activation function in machine learning which is used to add non-linearity in a machine learning model

WORK-FLOW: SUPPORT VECTOR MACHINE

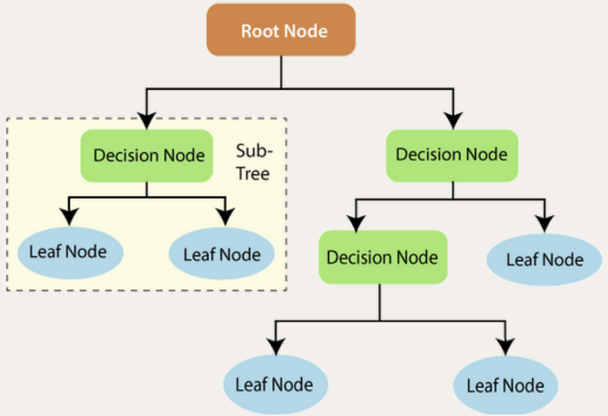
- From sklearn.svm we'll import SVM.
- If the dataset is linearly seperable, then no need to use kernel function. But our data is linearly inseparable.
- Therefore, first we have to convert the inseparable data to seperable data using kernel function.
- First, create a SVM object with kernel function = 'rbf'.
- Then, fit the training data to svm object.
- Now, try to adjust the values of C as 100.0, 1000.0, ... and check for better accuracy.
- Let's try using different kernel function, SVM object with kernel function = 'poly'.
- Now, try to adjust the values of C and try to obtain better accuracy.

KEYWORDS:

C = Tells the amount of misclassification that can be ignored.
rbf = radial bias kernel function
poly = polynomial kernel function
linearly seperable data = If you can draw a line or hyper plane that can separate those points into two classes.
linearly inseparable data = If you cannot draw a line or hyper plane that can separate those points into two classes.

WORK-FLOW: RANDOM FOREST CLASSIFIER

- From sklearn.ensemble we'll import RandomForestClassifier.
- Create a RandomForest object with 200 decision trees(n_estimators = 200).
- Now fit the training data to RandomForest object.
- This RandomForest model will now have outputs from 200 decision trees.
- Now the average of these 200 decision trees gives the final output.
- The combining of predictions to get final prediction is known as ensemble learning.
- Again train the model with different number of decision trees and check for better accuracy.



KEYWORDS:

n_estimators = number of decision trees.

WORK-FLOW: K NEAREST NEIGHBORS

- From sklearn.neighbors we'll import KNeighborsClassifier.
- First, create a knn object with k/ n_neighbors = 3, metric = 'minkowski', p = 4.
- Then, fit the training data to knn object.
- Now, try to increase the number of neighbors as 5, 7, 9 with metric = 'minkowski' and check for better accuracy results.
- For each testing data point, take k nearest neighbors and take the majority output as output.

KEYWORDS:

k/ n_neighbors = number of neighbors to be considered for testing.

minkowski distance= Distance measured between two points in N-dimensional space or in simple terms: a generalization of the Euclidean distance and the Manhattan distance.

p = 4 => order of the norm (variance, skewness, kurtosis, entropy)

WORK-FLOW: NAIVE BAYES(GAUSSIAN)

- From sklearn.naive_bayes we'll import GaussianNB.
- First, create a naive_bayes object.
- Then, fit the training data to naive_bayes object.
- In general, it is a probabilistic classifier, which means it predicts on the basis of the probability of an object.
- Convert the given dataset into frequency tables.
- Generate Likelihood table by finding the probabilities of given features.
- Now, use Bayes theorem to calculate the posterior probability.

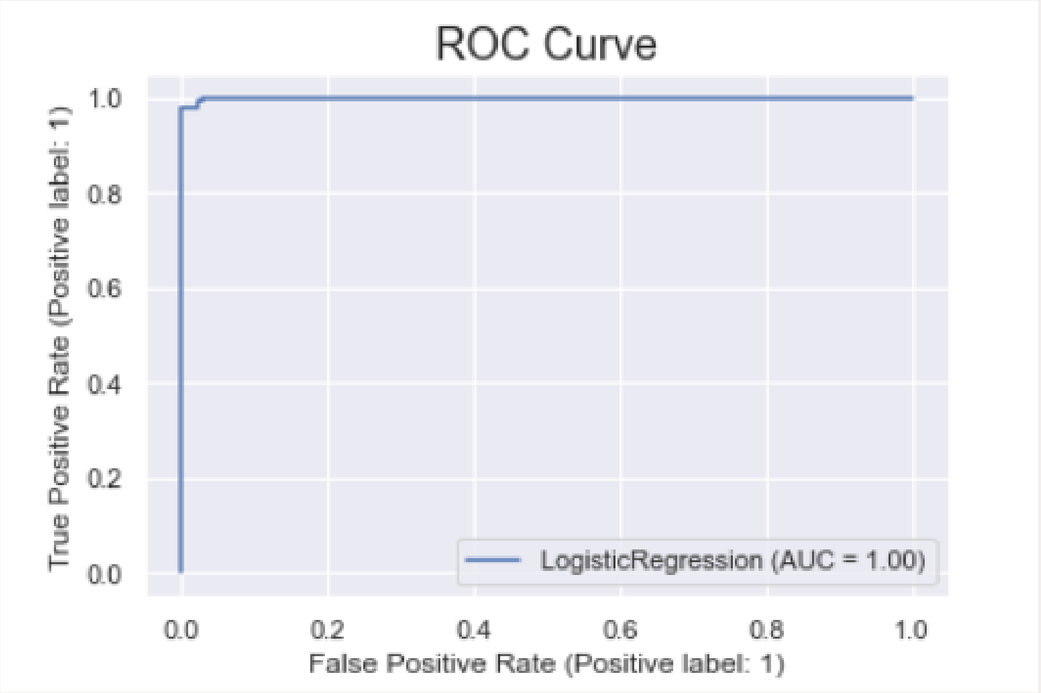
KEYWORDS:

Frequency tables: A table can quickly show us how many times each value appears.
Likelihood table: A table containing the data of the likelihood of events occurring.
Bayes theorem: Is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
Posterior probability: Is the revised or updated probability of an event occurring after taking into consideration new information

MODEL PERFORMANCE:

Logistic Regression:

```
Classification report:

              precision    recall  f1-score   support

           0       1.00      0.97      0.99       151
           1       0.97      1.00      0.98       124

    accuracy                           0.99       275
   macro avg       0.98      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275

Confusion Matrix
 [[147   4]
 [  0 124]]

Accuracy
 98.55 %
```



MODEL PERFORMANCE:

Support Vector Machine:

Classification report:

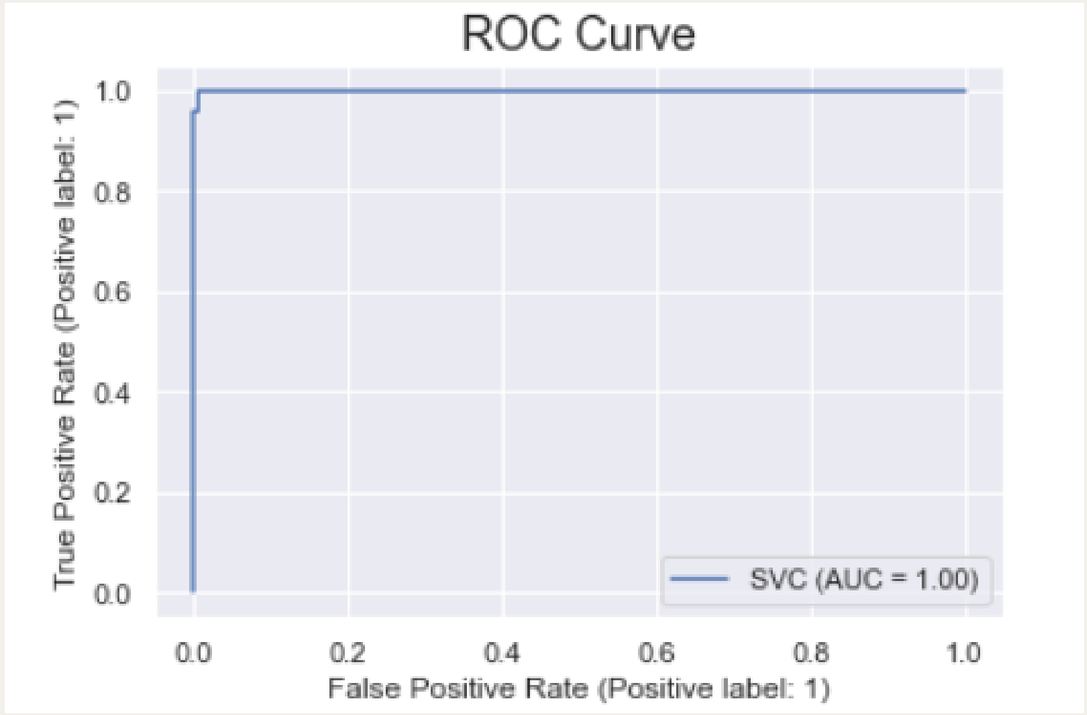
	precision	recall	f1-score	support
0	1.00	0.94	0.97	151
1	0.93	1.00	0.96	124
accuracy			0.97	275
macro avg	0.97	0.97	0.97	275
weighted avg	0.97	0.97	0.97	275

Confusion Matrix

```
[[142   9]
 [  0 124]]
```

Accuracy

96.73 %



MODEL PERFORMANCE:

Random Forest Classifier:

Classification report:

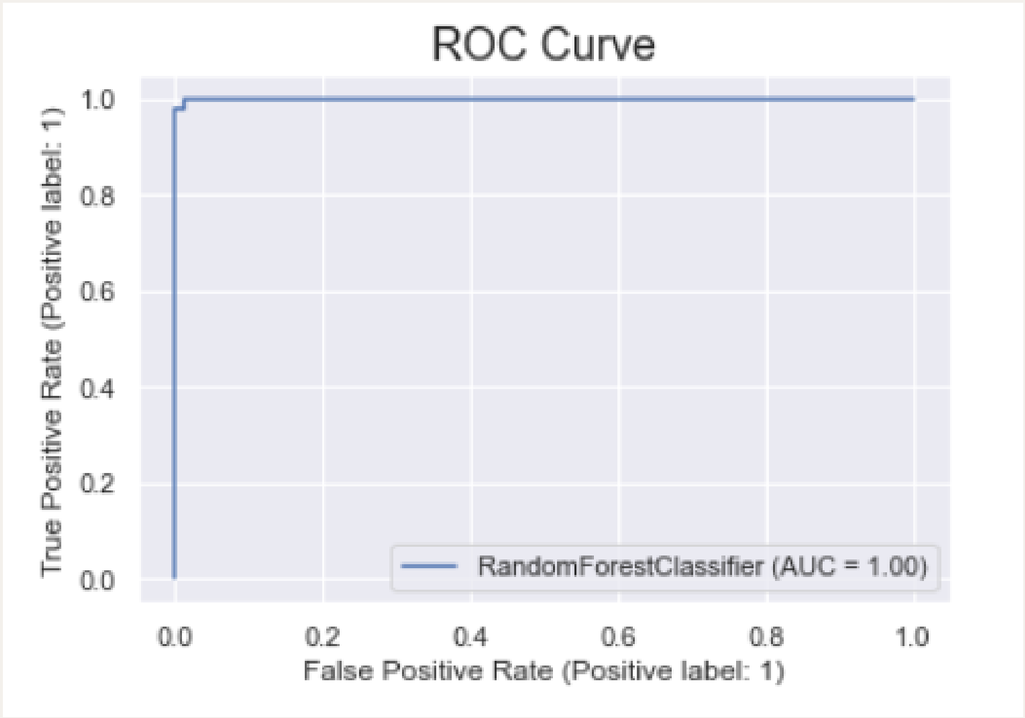
	precision	recall	f1-score	support
0	0.99	0.99	0.99	151
1	0.98	0.98	0.98	124
accuracy			0.99	275
macro avg	0.99	0.99	0.99	275
weighted avg	0.99	0.99	0.99	275

Confusion Matrix

```
[[149   2]
 [  2 122]]
```

Accuracy

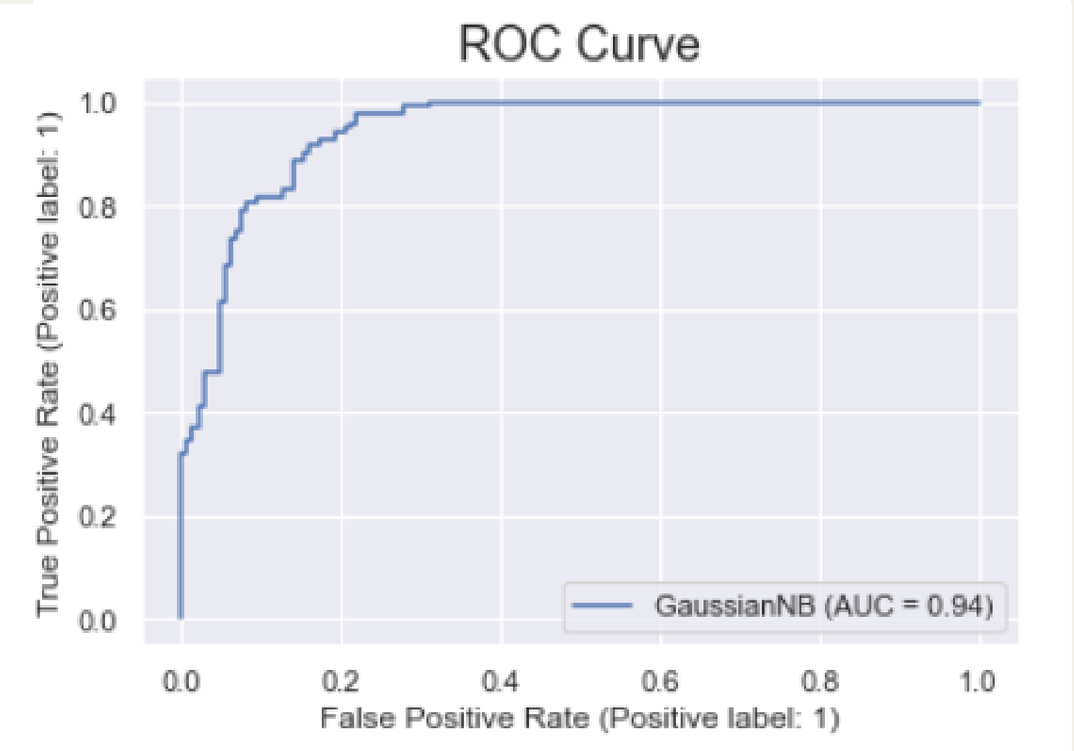
98.55 %



MODEL PERFORMANCE:

Bayes Theorem:

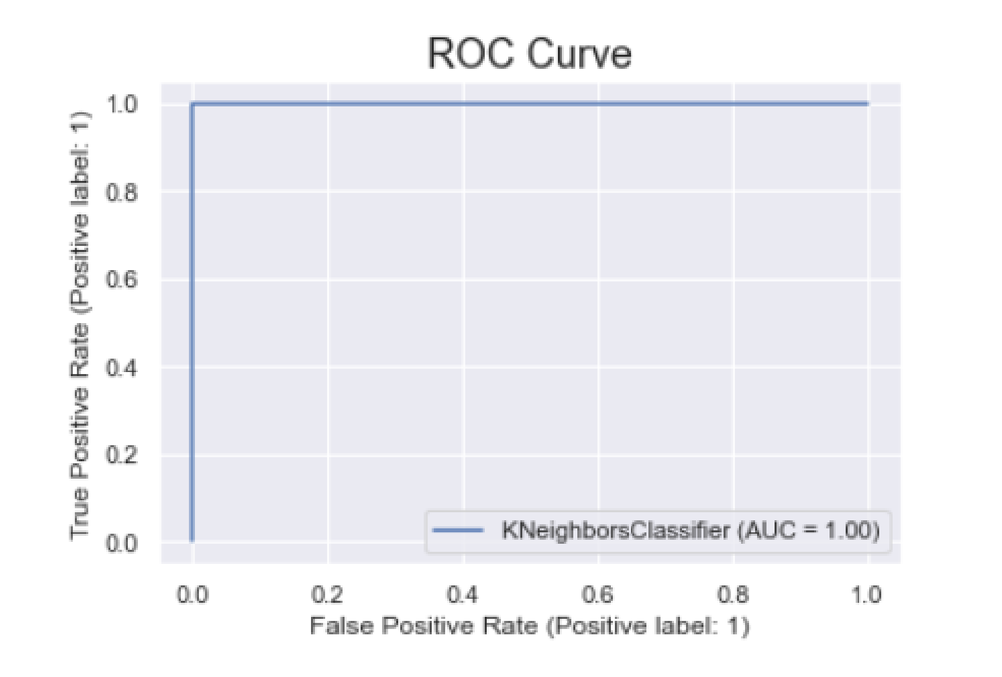
```
Classification report:

              precision    recall  f1-score   support

           0       0.86      0.87      0.87       151
           1       0.84      0.82      0.83       124

    accuracy                           0.85       275
   macro avg       0.85      0.85      0.85       275
weighted avg       0.85      0.85      0.85       275

Confusion Matrix
 [[132  19]
 [ 22 102]]

Accuracy
 85.09 %
```



MODEL PERFORMANCE:

K Nearest Neighbors:

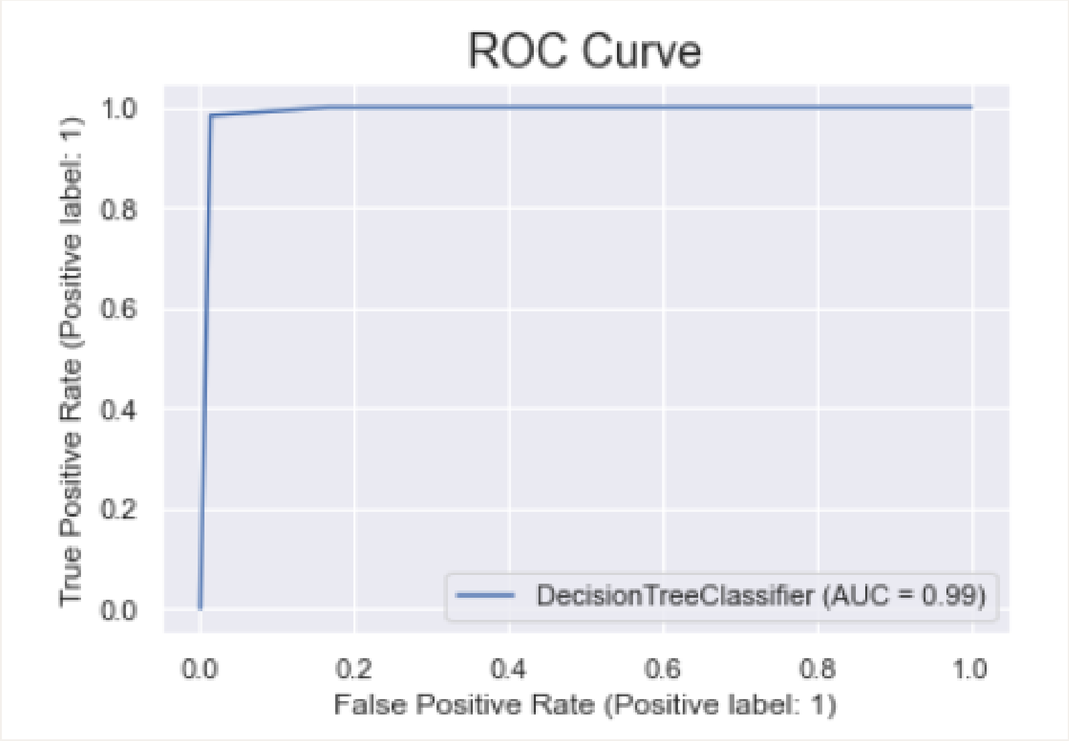
```
Classification report:

              precision    recall  f1-score   support

           0       1.00      0.99      1.00       151
           1       0.99      1.00      1.00       124

    accuracy                           1.00       275
   macro avg       1.00      1.00      1.00       275
weighted avg       1.00      1.00      1.00       275

Confusion Matrix
 [[150   1]
 [  0 124]]

Accuracy
 99.64 %
```



MODEL PERFORMANCE:

Decision Trees (Gini Index algorithm):

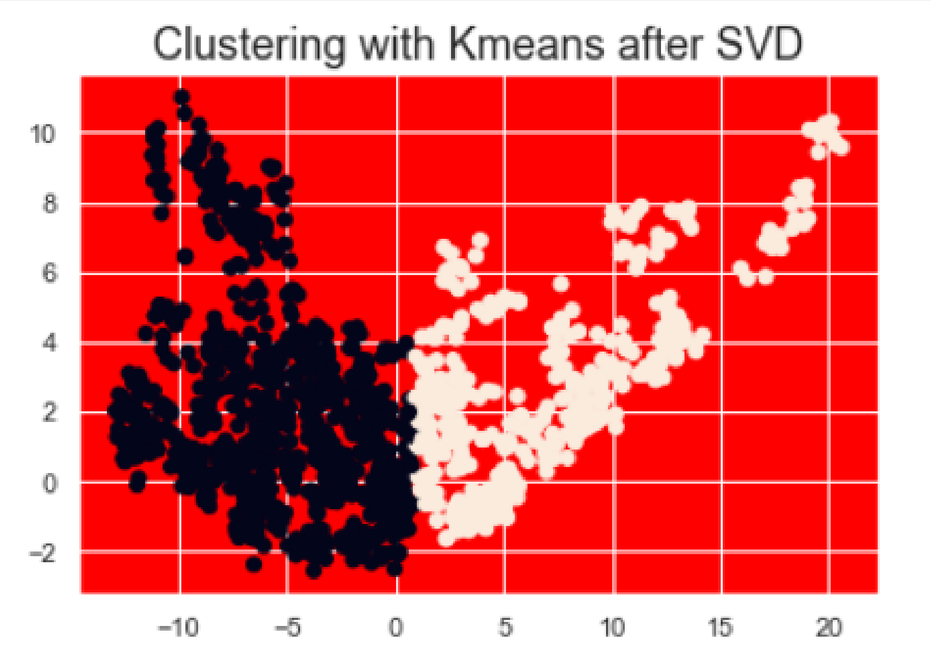
```
Classification report:

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       151
           1       0.98      0.98      0.98       124

    accuracy                           0.99       275
   macro avg       0.99      0.99      0.99       275
weighted avg       0.99      0.99      0.99       275

Confusion Matrix
 [[149   2]
 [  2 122]]

Accuracy
 98.55 %
```



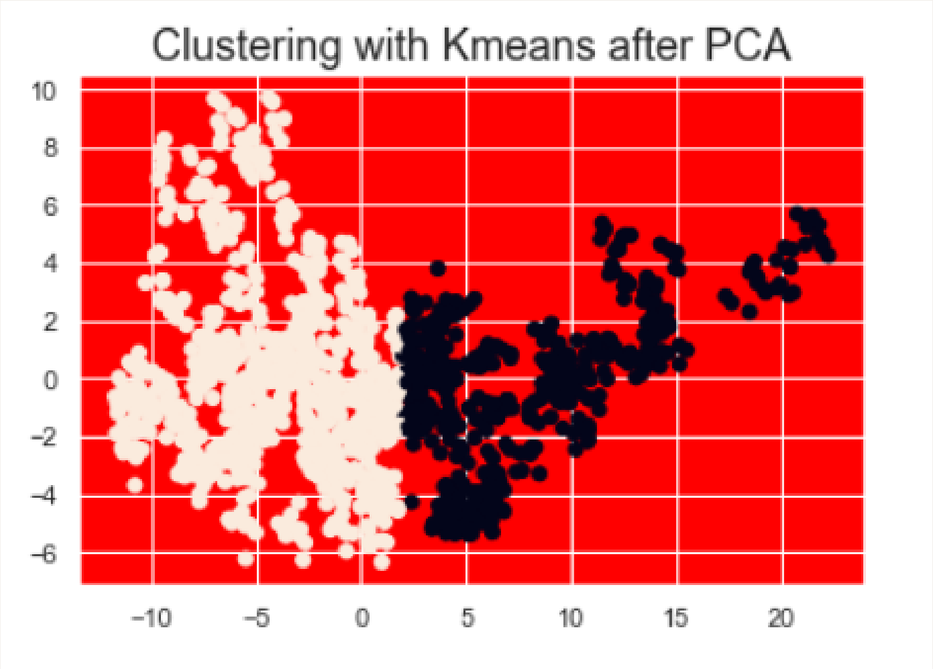
MODEL PERFORMANCE:

K-Means Clustering:

K-Means with svd:



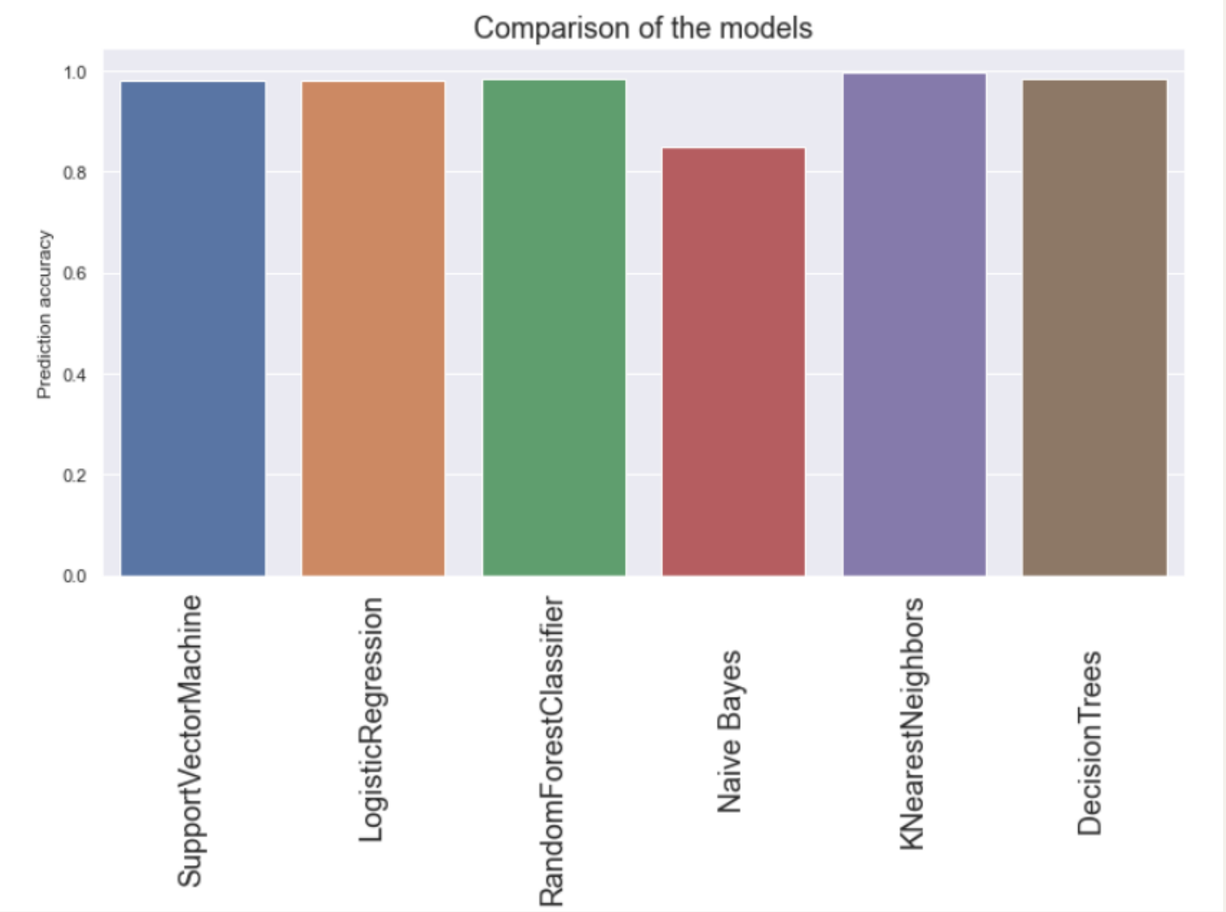
K-Means after PCA



COMPARISON:

Algorithm	Accuracy
SupportVectorMachine	98.18 %
LogisticRegression	98.18 %
RandomForestClassifier	98.55 %
Naive Bayes	85.09 %
KNearestNeighbors	99.64 %
DecisionTrees	98.55 %

Due to less amount of data, almost all the algorithms worked very well. But, KNN algorithm is having the upper hand in our scenario. Whereas, Naive Bayes is having the least accuracy but not a bad one.



FUTURE SCOPE OF WORK:

- The similar approach may be built for the remaining Indian currency notes as well as currency notes from other countries in the future, making mobile apps helpful for both normal people and people with visual impairments. The app's UI may also be further customised to meet user needs.
- Our future scope will be concentrated on fast and more accurate fake currency detection using advanced image processing techniques.
- This system can be further implemented for foreign currencies like Dollars, Euros, Taka, etc. as a future scope

THANK YOU!